



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	§	Docket No.:	35399.42
Heeloo Chung, et al.	§		
	§	Customer No.	27683
Serial No.: 09/990,753	§		
	§	Group Art Unit:	2663
Filed: November 16, 2001	§		
	§	Examiner:	Derrick W. Ferris
For: High Efficiency Data Buffering	§		
In a Computer Network Device	§		

DECLARATION UNDER 37 C.F.R. § 1.131

I, Heeloo Chung, declare and say that:

1. I am a joint inventor of the subject matter disclosed and claimed in the above-identified application.
2. At all times set forth herein, I was employed at Force10 Networks, Inc. ("Force10"), Milpitas, California. Force10 is the assignee of the above-identified application.
3. Prior to July 18, 2001, I was part of a Force10 team who reduced the subject matter of the present invention to practice in Milpitas, California, in an Application Specific Integrated Circuit (ASIC) we designed, codenamed "Cougar."
4. Exhibit A contains several pages excerpted from a Cougar ASIC Internal Design Specification, numbered manually for reference in this declaration. I was one of the authors of this document. Although minor details of the Specification regarding control register definitions not important to this invention were updated just subsequent to July 18, 2001, the subject matter appearing on the pages excerpted in Exhibit A was placed in the Specification prior to July 18, 2001.
5. Pages 1-3 of Exhibit A show the general architecture of a network device (the "Cyclone switch system") with the Cougar ASIC serving as a line card buffer manager between an ingress/egress ASIC (codenamed "Tiger") and a switch fabric comprising switch fabric ASICs (codenamed "Simba"), with switch fabric transfers controlled by a scheduler ASIC (codenamed "Panda").

Appl. No. 09/990,753
Customer No. 27683

6. Pages 4-9 of Exhibit A show details of the buffer manager implemented on the Cougar ASIC, with data managed in "chunks" analogous to the "trunks" disclosed in the present patent application. The Cougar ASIC buffer manager implements the buffer manager features claimed in the present application.

7. Exhibits B and C are copies of e-mails that I sent on the first and second days of testing of the Cougar ASIC, with dates redacted. These e-mails report our activities, during which I was personally present, to bring up the Cougar ASIC prior to July 18, 2001.

8. A "port pipe" as we use the term is a data flow stream between a Cougar and the switch fabric. According to Exhibit C, on the second day of testing we had a network device operating in a loop including an ingress and an egress Cougar, with over 1 billion error-free packets transmitted.

9. Based on my recollection and on Exhibits B and C, I am confident that we had reduced the Cougar ASIC to practice, and a network device containing Cougar ASICs to practice, prior to July 18, 2001.

I declare that all statements made herein of my knowledge are true and that all statements made on information and belief are believed to be true and that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.


Heeloo Chung

Date: 1-19-2006

THIS PAGE BLANK (USPTO)

Exhibit A

3.1 The Architecture Overview

Ingress Data Flow Diagram

The diagram illustrates the ingress data flow through a switch architecture. The central component is the **Cougar** switch, which contains several internal units:

- Buffer/Queue Management Unit**: Located at the top of the Cougar switch, connected to **Buffer Memory** and **Queue Memory** via bidirectional control paths.
- Front End C-Port Interface**: Receives data from the **Tiger** (external system) via a data path.
- Packet Processing Unit**: Receives data from the Front End C-Port Interface via a data path and sends control signals to the Buffer/Queue Management Unit.
- Output Organizer**: Receives data from the Packet Processing Unit via a data path and sends control signals to the Buffer/Queue Management Unit.
- Backplane C-Port Interface**: Receives data from the Output Organizer via a data path and sends control signals to the Buffer/Queue Management Unit.
- PCI Interface**: Connected to the **CPU** via a bidirectional control path.
- Scheduler Interface**: Connected to the **Panda** (external system) via a bidirectional control path.

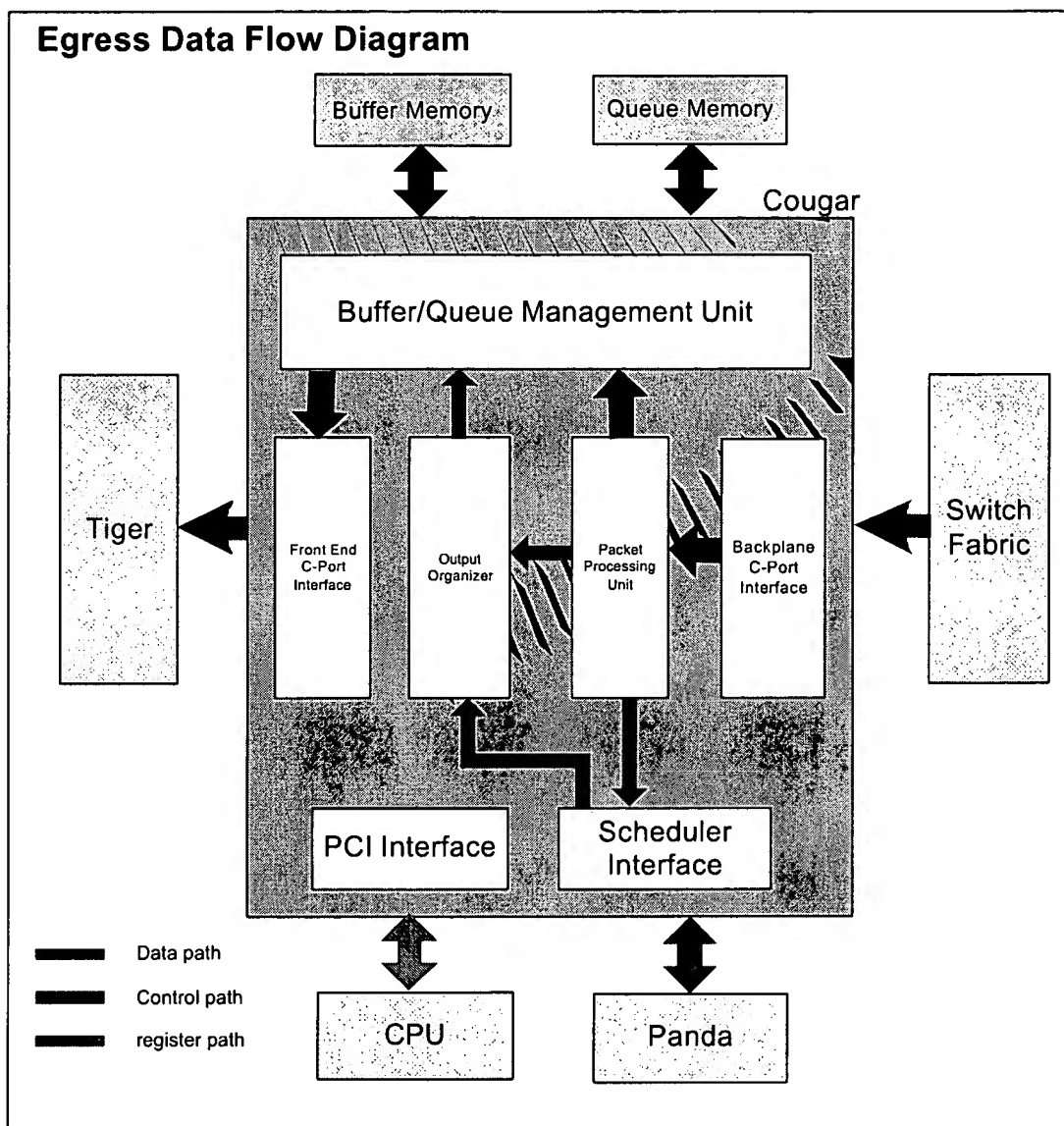
External components and their connections:

- Tiger**: External system connected to the Front End C-Port Interface via a data path.
- Switch Fabric**: Connected to the Backplane C-Port Interface via a data path.
- Buffer Memory**: Connected to the Buffer/Queue Management Unit via a bidirectional control path.
- Queue Memory**: Connected to the Buffer/Queue Management Unit via a bidirectional control path.
- CPU**: Connected to the PCI Interface via a bidirectional control path.
- Panda**: Connected to the Scheduler Interface via a bidirectional control path.

Legend:

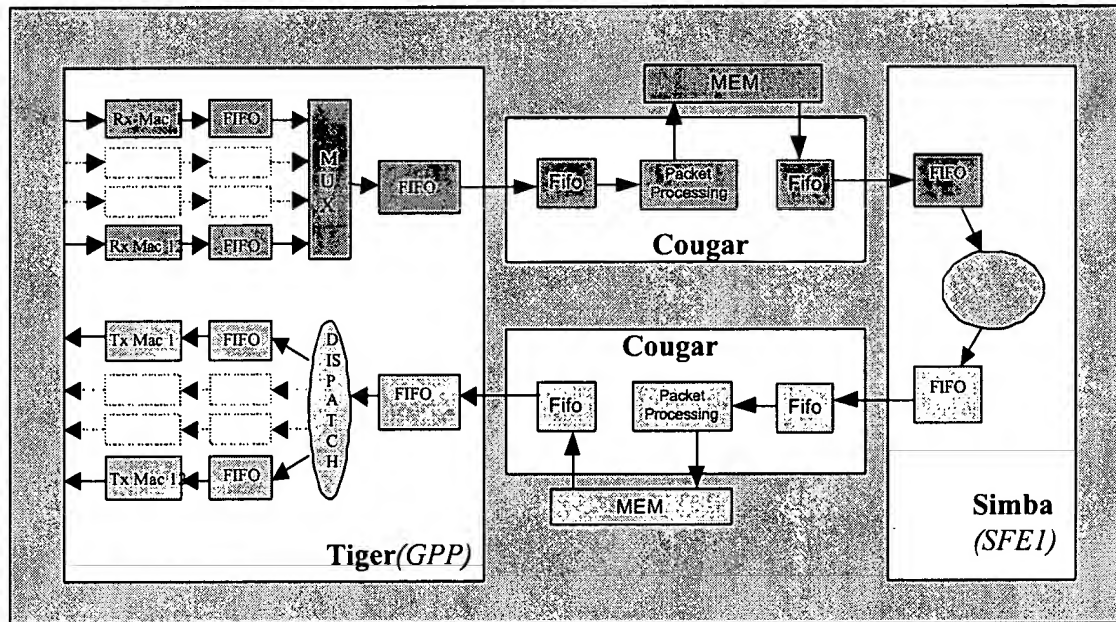
- Data path**: Represented by a solid black arrow.
- Control path**: Represented by a thick black arrow.
- register path**: Represented by a dashed black arrow.

BEST AVAILABLE COPY



3.2 System level Data Flow

Cougar chip is used multiple times in the Cyclone system. It is used in both the receive data path and transmit data path. For each Tiger used in the system, two Cougar will be used, one for receive and one for transmit. A data packet comes into the system will go through two Cougars. See the following basic data flow diagram. In the current design, each Cougar can handle up to 12 Gbps of input data traffic.



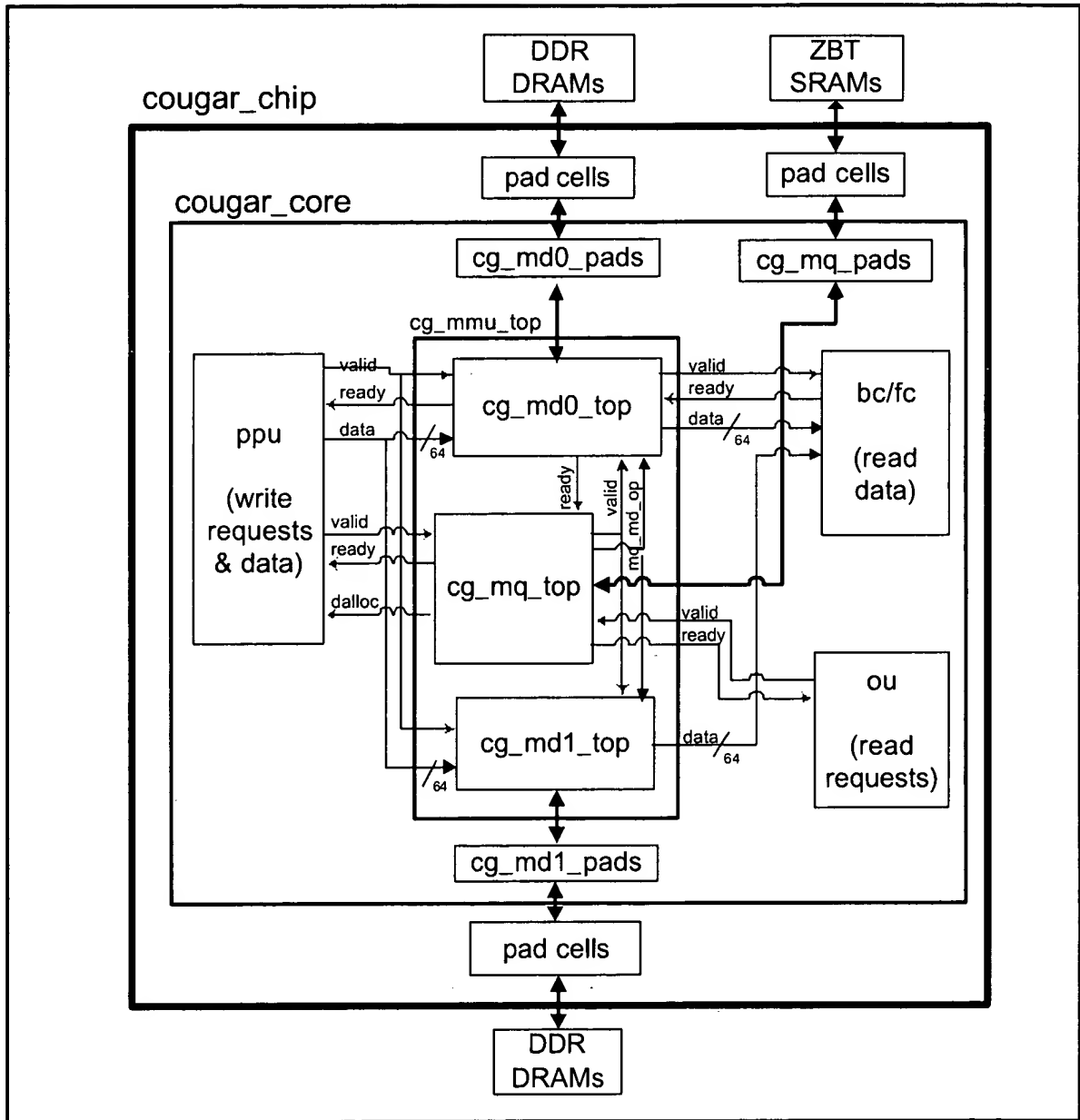
3.3 Chip Level Interfaces

Cougar has 6 external interfaces, one Tiger C-port interface, one back plane C-port interface, two memory interfaces, one CPU interface, and one scheduler interface. The Tiger C-port interface is used in between the Tiger and Cougar. The back plane C-port interface is in between the Cougar and the back plane switch fabric. Two memory interfaces are used for data storing and forwarding. Panda is the scheduler for the back plan switch fabric. Cougar will use the scheduler interface to communicate it. During the system power up, PCI interface will be used to initialize the chip. This interface can also be use for monitoring the status during the normal operation. All interfaces are described in detail in the following sections.

3.3.1 Tiger C-Port Interface

C-Port interface is a DDR (double data rate) bus interface, runs at 160MHz. Since each Tiger has to handle 12 ports at the same time, data packets will be multiplexed in and out of the C-port in TDM fashion. C-port has two modes. It works as input only interface in the ingress path, and output only on the egress path. Most of signals are bi-directional I/O's. It consists of the following signals:

4.4 Memory Management Unit



4.4.1 Overview

4.4.1.1 Terminology

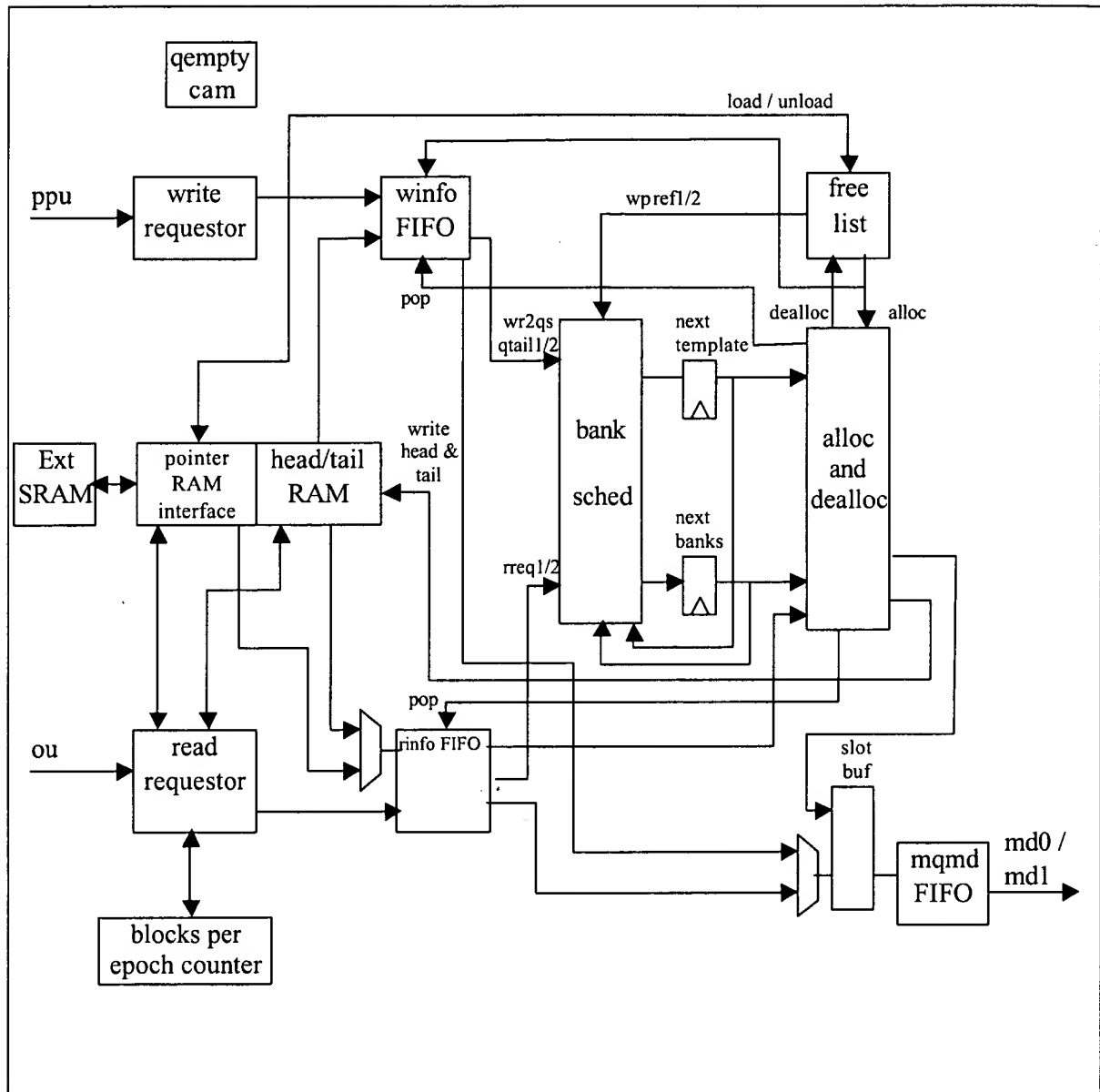
- **Chunk** 128-byte data storage unit. Corresponds to a burst of 4 words to/from the DDR DRAMs. A chunk is made of 8 blocks.
- **Block** 16-byte data unit written by the ppu or read by the fc or bc.
- **Template** A sequence of commands to the DDR DRAMs that accesses four different banks. Made of four slots.
- **Slot** Access to a single bank in a template
- **ECC** Error correcting code
- **Header ECC** The ECC bits for the 96 bit (12 byte) F10 header in data memory. Occupies 8 bits.
- **Pointer** A single location in external queue SRAM that contains a link to the next node, the packet length and ECC bits.

4.4.1.2 Structure

The mmu block consists of six core-level blocks, as shown on the diagram above. The blocks are:

1. **cg_mq_top** Receives write requests from the ppu and read requests from the ou. Groups the reads and writes into DRAM templates with two reads and two writes. Allocates and deallocates the memory chunks from the freelists. Maintains FIFO head and tail information for up to 256 unicast and 128 multicast queues.
2. **cg_mq_pads** Registers that are placed close to the ZBT SRAM I/O cells on the layout. Includes the ECC generator and checker for the pointers.
3. **cg_md0_top** The data unit for one half of the 256 DDR DRAM pins. Contains read and write FIFOs and a DRAM controller sequencer. Also contains a header ECC checker/corrector for half the DRAM pins.
4. **cg_md1_top** The data unit for the second half of the 256 DDR DRAM pins. Contains read and write FIFOs and a DRAM controller sequencer. Also contains a header ECC checker/corrector for half the DRAM pins.
5. **cg_md0_pads** Registers that are placed close to the DDR DRAM I/O cells on the layout. Includes the DDR read data strobe delay lines and FIFOs.
6. **cg_md1_pads** Registers that are placed close to the DDR DRAM I/O cells on the layout. Includes the DDR read data strobe delay lines and FIFOs.

4.4.3.2 cg_mq_top Block Diagram



4.4.3.3 DDR DRAM Basics

DDR (Double Data Rate) DRAMs are Dynamic memory chips that are internally organized as 4 banks by R Rows by C columns. Each Column is a data word, either 16 bits or 32 bits per chip for the types that are interesting for this unit. There is an address and command bus that is sent to the DRAM synchronized by a clock. The address and command pins are single data rate (SDR). The address is sent in two parts – the row address first, then the column address.

The sequence of events needed to read from the DRAM is:

1. Wait until tRP has been satisfied for the desired bank. Send NOPs until this timing parameter is met.
2. Activate the bank - send the ACT command along with the desired bank number and row address.
3. Wait for tRCD clocks
4. Send the READAP (read with auto precharge) along with the column address and the bank number.
5. Wait until the read data strobe is returned by the DRAM.
6. Capture the read data on both positive and negative edges of the data strobe.

The sequence needed to write to the DRAM.

1. Wait until tRP has been satisfied for the desired bank. Send NOPs until this timing parameter is met.
2. Activate the bank - send the ACT command along with the desired bank number and row address.
3. Wait for tRCD clocks
4. Send the WRITEAP (write with auto precharge) along with the column address and the bank number.
5. Drive the data strobe outputs low during the preamble.
6. Drive the write data onto the DQ outputs and toggle the data strobe. The write data strobe is centered around the write data. Data is transferred on both edges of the data strobe.
7. When the entire burst has been output, drive the data strobes low during the postamble.
8. Drive the data strobes to tri-state.

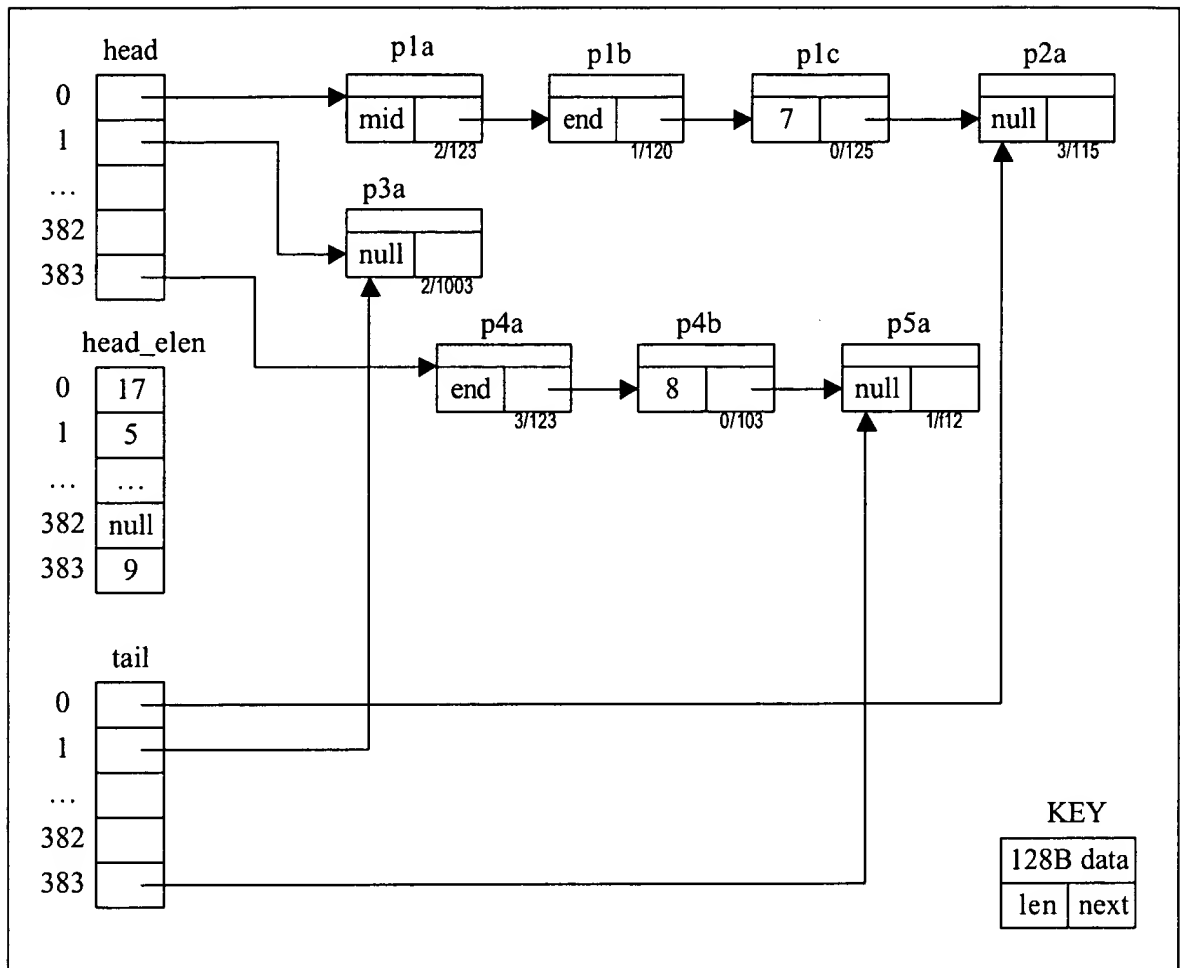
Operations for each bank can happen in parallel. We use this fact to interleave all four banks within one 13-cycle template to get two read bursts and two write bursts every 13 cycles (with best bank assignment). With worst bank assignment, up to 5 *spacer cycles* (see below) must be added, extending the template time to 18 cycles.

4.4.3.4 Queue Structure

The mq unit supports 256 unicast and 128 multicast queues. Each queue is a FIFO. For each queue there is:

- A head pointer (head[q]), which points to the first chunk in the first packet in the queue.
- A tail pointer (tail[q], which points to the last chunk in the last packet in the queue.
- A head length (head_len[q]), which indicates the size in blocks of the first packet in the queue.

Conceptually, there are three SRAM arrays on chip that hold the queue start and end information. Physically, all three arrays are kept in a single 1-port SRAM.



4.4.3.5 Read Requests

Read requests are sent by the ou. The ou requests a queue to read by setting ou_mq_mcast, ou_mq_qid, and ou_mq_last_mcast and setting ou_mq_valid high. The mq accepts the read request by setting mq_ou_ready high. Read requests for the packets pointed to by the head[q] register for the queue will be pushed to the rinfo_fifo. The linked list for the queue is traversed until any of the following occurs to terminate reading the queue:

1. The end of the queue is reached, as indicated by reaching the node pointed to by the tmp_read_tail register. At the beginning of traversing a queue, the tail[q] value is saved to tmp_read_tail.
2. An ou_mq_epoc_stop is received and the end of a packet is seen.
3. The blocks_per_epoch counter indicates that the next packet on the list will not fit into the current epoch. This check does not happen for the first packet in the epoch.
4. If the chip is in egress mode and a multicast request is received, when the end of the first packet has been processed.

During the process of walking the linked list, each chunk is automatically deallocated (returned to the free list) except for the following two conditions:

1. multicast request and ou_mq_last_mcast was low.
2. Packet generator mode is enabled.

4.4.3.6 Write Requests

Write requests are sent by the ppu. An individual write request occurs for each chunk in the packet. Packet boundaries are indicated by the pp_mq_header and pp_mq_tail signals, which are sent along with pp_mq_mcast and pp_mq_qid when pp_mq_valid is high to initiate the request. The mq accepts the request by setting mq_pp_ready high.

For the first chunk in the packet, the size of the entire packet is indicated on pp_mq_psize, in blocks. For all subsequent chunks in the packet, pp_mq_psize should have a range from 1 to 8 blocks. The middle chunks should have a size of 8, and the last chunk a size from 1 to 8, except for one-chunk packets. One-chunk packets must have a size from 5 to 8 blocks inclusive.

The register head_elen[q] is checked for the requested queue number. If the queue is empty, then the head[q] and head_elen[q] registers will need to be written after the blocks in the packet have been allocated. If the selected queue is not empty, then the tail[q] register is read and pushed to the winfo_fifo. The qtail bank number will be used by the bank scheduler to ensure that two chunks on a single queue never have the same bank number.

All the chunks in write packet are first built off-line and linked together before being appended to the queue. The entire packet is then appended in one atomic operation. The registers tmp_write_head, tmp_write_head_elen and tmp_write_tail are used to hold the write packet while it is being built.

4.4.3.7 Bank Scheduler

The bank scheduler combines two read chunk requests with two write chunk requests to form a template to be executed by the md0/md1 units. The following templates are supported:

Template	0	1	2	3	4	5	6	7	8	9	10	11	12
RaRbWcWd	ACTa	NOP	ACTb	RDa	NOP	RDb	NOP	ACTc	NOP	ACTd	WRc	NOP	WRd
WaWbRcRd	ACTa	NOP	ACTb	WRa	NOP	WRb	ACTc	NOP	ACTd	RDC	NOP	RDd	NOP

The inputs to the bank scheduler are:

1. rreq1 & rreq2: Zero, one or two read requests – bank numbers
2. qtail1 & qtail2: Zero, one or two write requests – bank numbers for the previous nodes that the newly allocated nodes will link to.
3. wr2qs: If there are two write requests, are the requests for a single queue (need to be chained together) or for two different queues.

From: Heeloo Chung
Sent:
To: Michael Laudon; Force10-MRandall-staff; Force10-managers; Force10-Ripley; PK Dubey
Cc: Force10-asics-grp
Subject: RE: Latest Ripley Status

Day 1 cougar bringup status:

1. all three PLL's are working.
2. PCI register read/write are working
3. PCI read/write SRAM (link-list pointer) works
4. PCI read/write SDRAM (buffer memory) works
5. sending 6 packets from Tiger into Ingress Cougar (bypass Cheetah), intercept the packets from Ingress Cougar buffer memory through the PCI. Packets are looking good.

Issues:

1. We have some problem setting up the Lynx in fake-panda mode.

That is why we can not move the packets out of Ingress Cougar.

We will work on this tomorrow.

2. Some diag/driver problems need to be clean up.

Right now all the tests we have done are through semi-manual mode.

There were WAY TOO MUCH manual typing. We need to automate the tests a little better.

3. A bad crystal on board #9 need to be fixed.

Tremendous progress for the 1st day. Thanks to all the people involved on this bringup effort.

-Heeloo

> -----Original Message-----
> **From:** Michael Laudon
> **Sent:**
> **To:** Force10-MRandall-staff; Force10-managers;
> Force10-Ripley; PK Dubey

> **Subject:** Latest Ripley Status
>
>
> Cougar PLLs tested, all three look good after a quick look.
>
> Cougar SRAM and SDRAM accesses via PCI are functioning. Only
> a few memory accesses
> were tested, but they worked without error. Testing on the
> memory interface is very preliminary
> and we won't be able to sign off on a fully verified memory
> interface until Glenn Poole completes
> his testing several weeks later and the systems group
> performs DVT on the full Ripley PCB.
>
> Regards
> --Mike
>
> *Michael Laudon*
> *Engineering Manager*
> *Force10 Networks*
> *Tel: (408)571-3510*
> *Fax: (408)571-3550*
> **www.force10networks.com**
>

From: Heeloo Chung
Sent:
To: Force10-eng-grp
Cc: PK Dubey
Subject: OVER 1 BILLION PACKETS !!!!!

Here we come, Ripley first port pipe is alive:

IXIA->Tiger-IngressCougar (with Lynx)->BackplaneSerdes->EgressCougar->Tiger->IXIA

We are setting the IXIA in a infinite loop. So far we have **over 1 billion packets** through the pipe with **NO ERROR** and no dropping.

-Heeloo